НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО»

Теплоенергетичний факультет

Кафедра автоматизації проектування енергетичних процесів і систем

До захисту допущено

Завідувач кафедри

_____ О.В. Коваль
(підпис) (ініціали, прізвище)

“___”_____2020р.

ДИПЛОМНА РОБОТА

на здобуття ступеня бакалавра

з напряму підготовки 121 “ Інженерія програмного забезпечення “

на тему Лінгвістичне забезпечення опису сцени проведення гідроакустичних експериментів

Виконав : студент 4 курсу, групи ТІ-61

_____ Єдинак Юліан Ігорович _____ _____
(прізвище, ім’я, по батькові) (підпис)

Керівник ___ старший викладач, Гайдаржи В. І. _____ _____
(посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Рецензент ___ доцент Побіровський Ю. М. _____ _____
(посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2020

Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”

Факультет теплоенергетичний

Кафедра автоматизації проектування енергетичних процесів і систем

Рівень вищої освіти перший , бакалаврський

Напрям підготовки 121 “ Інженерія програмного забезпечення “

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ О.В. Коваль
(підпис)
”___”_____2020р.

ЗАВДАННЯ

на дипломну роботу студенту

_____Єдинаку Юліану Ігоровичу_____
(прізвище, ім’я, по батькові)

1. Тема роботи Лінгвістичне забезпечення опису сцени проведення гідроакустичних експериментів
керівник роботи Гайдаржи Володимир Іванович, старший викладач
(прізвище, ім’я, по батькові науковий ступінь, вчене звання)

затверджена наказом вищого навчального закладу від ”__”___ 201_р. № __

2. Строк подання студентом роботи _____

3. Вихідні дані до роботи Програма реалізована на мові Kotlin, для платформи Windows

4.Зміст розрахунково-пояснювальної записки (перелік завдань, які потрібно розробити) Розробити мову для опису експерименту, реалізувати транслятор для розробленої мови, створити середу для зручної роботи з описом експерименту, автоматизувати друк шаблонного тексту

5. Перелік ілюстративного матеріалу Титульний лист, постановка задачі, синтаксичний підхід до опису складних структур, фрагмент граматики мови експерименту, структура транслятора, приклад опису експерименту, результуючий JSON, генерація опису експерименту, додаткове редагування опису експерименту, використані технології, висновки

6. Дата видачі завдання ”11” __ жовтня ___2019р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітки
1.	Вивчення та аналіз задачі		
2	Розробка архітектури та загальної структури системи		
3.	Розробка структур окремих підсистем		
4.	Програмна реалізація системи		
5.	Оформлення пояснювальної записки		
6.	Захист програмного продукту		
7.	Передзахист		
8.	Захист		

Студент _____ _____Єдинак Ю. І._____
(підпис) (прізвище та ініціали,)

Керівник роботи _____ _____Гайдаржи В. І_____
(підпис) (прізвище та ініціали,)

АНОТАЦІЯ

Метою виконаної роботи було створення зручної, простої та ефективної системи для лінгвістичного забезпечення опису сцени проведення гідроакустичних експериментів.

У результаті було розроблено мову опису експерименту, транслятор до цієї мови, створено графічний інтерфейс, який дозволяє легко працювати з текстом опису експерименту, транслювати його у вихідний формат, а також генерувати опис за необхідності.

При виборі методів для реалізації транслятора, увага приділялася легкості розширення мови опису, через що був обраний метод висхідного розбору для синтаксичного аналізу та метод регулярних виразів для лексичного.

Представлена дипломна робота займає 51 сторінку по своєму об’єму та включає 23 рисунка, 6 посилань і 3 додатки.

ABSTRACT

The purpose of the work was to create a convenient, simple and efficient system for linguistic support of the description of the scene of hydroacoustic experiments.

As a result, the language of the experiment description was developed, a translator to this language, a graphical interface was created, which allows you to easily work with the text of the experiment description, translate it into the original format, and generate a description if necessary.

When choosing methods for translator implementation, attention was paid to the ease of expanding the description language, which is why the ascending parsing method for parsing and the regular expression method for lexical analysis were chosen.

The presented thesis occupies 51 pages in its volume and includes 23 figures, 6 references and 3 appendices.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

JSON - JavaScript object notation

SQL - structured query language

JVM - Java virtual machine

POJO - plain old java object

DSL - domain specific language

CSS - cascading style sheets

ТТХ - техніко-технічні характеристики

JRE - Java runtime environment

ВСТУП

Вже було розроблено багато різних інструментів для опису інформації, починаючи від усної мови і закінчуючи різного роду візуалізаціями, двовимірними діаграммами або навіть тривимірними проекціями. Проте у науковій середі основними засобами опису інформації є текст або діаграми.

У випадку інформації про гідроакустичний експеримент, необхідно описати його структуру, яка складається з сцени, морського об’єкту та сценарію руху об’єкту. Ця структура має декілька рівнів вкладеності, яку насправді можливо описати за допомогою діаграми, але необхідність кожного разу створювати окрему діаграму для опису експерименту може ускладнити та сповільнити створення експериментів. Саме по цій причині був обраний текстовий спосіб опису, який є найбільш простим і зрозумілим для більшості.

Текст дозволяє описувати структури даних з необмеженою кількістю рівнів вкладеності. Всі сучасні інформаційні системи використовують текст для опису команд до компьютера. Через це і усі структури даних і їх організація описуються цим способом.

Через велику кількість вже створених систем, їм необхідний спосіб за допомогою якого було б можливо організувати спілкування між ними. Існує багато уніфікованих форматів за якими системи обмінюються інформацією. JSON є основним основним, по розповсюдженості, способом опису структур даних, його використовують різного роду Rest-системи для відправки складної, за своєю структурою, інформації. Іншим способом описати складні структури є формат XML, він у більшості випадків використовується для розмітки і описує структури у більш простішому для сприйняття форматі ніж формат JSON.

Для користувачів інформаційних систем більш зрозумілим є графічний спосіб опису. Через це графічні інтерфейси і набули популярності, це дозволяє

користувачам не думати про організацію даних всередині системи, а розробникам створювати зручні та зрозумілі інтерфейси, якими можуть користуватися люди, які вперше їх бачать.

Форма реєстрації є найбільш простим прикладом опису структури даних за допомогою графічного інтерфейсу. Вона дозволяє користувачу створити внутрішнє представлення структури користувача у системі і заповнити його поля, яка не буде мати рівнів вкладеності, але дозволить без знання програмування взаємодіяти з даними системи. Також зазвичай форми реєстрації мають функціонал для спрощення введення даних користувачем, наприклад перевірка полів на коректність під час їх заповнення, яка сповіщає користувача про те що він допустив помилку при написанні адреси або обрав занадто простий пароль. Ці сповіщення відбуваються за допомогою тієї ж форми, зазвичай невірні поля виділяються червоним кольором і під полем для введення з’являється текст сповіщення. Також структура багатьох форм дозволяє користувачу не плутати формати даних, так як кожне поле для введення тексту підписано, а інколи, у випадках як наприклад коли необхідно ввести дату, додається випадаючий список, або навіть окреме вікно вибору дати, яке повністю гарантує коректність введення дати. Усі ці спрощення дозволяють користувачу не запам’ятовувати кількість, формат або порядок введення даних. І навіть інколи поле саме редагує себе щоб отримати вірний формат, наприклад у випадку коли код країни, під час вводу номера телефону, не треба вводити, поле може саме видаляти перші два символа, коли бачить що ці числа співпадають з кодом, а користувач намагається ще ввести числа. Така велика кількість спрощень, яка прискорює роботу з даними, робить графічні інтерфейси найбільш популярними серед домінуючої більшості користувачів. Через це у сучасних реаліях була сформована тенденція необхідності у графічних інтерфейсах, що призводить до масового створення такого роду інтерфейсів.

Метою представленої дипломної роботи було створення інтуїтивної системи лінгвістичного забезпечення для опису експерименту з гідроакустики. Під

інтуїтивністю розуміється простота, зрозумілість та прозорість у керуванні системою, навіть без досвіду роботи з нею. Також від користувача не мають вимагатися глибокі знання у програмуванні. Через усі ці необхідності очевидною є потреба у окремому редакторі коду, який дозволив би зручно і просто користувачу працювати з текстом опису.

На сьогоднішній день вже розроблено багато програм, які слугують цій цілі, до них відносять такі середовища програмування, як IntelliJ IDEA, Eclipse, Web Storm, які підтримують окремий, часто з можливістю розширення, набір мов програмування. Вони валідують вхідний текст, виправляючи все помилки у написанні програмістом програмі ще до того як вона буде скомпільована. Великим пл.сом таких середовищ є можливість їх налаштовувати. Можливо змінювати шрифти, розміри символів, цвітові набори, способи форматування тексту, влаштовуючи його під потреби загального стилю серед розробників, навіть змінювати форму інтерфейсу керування, приховуючи або додаючи різні панелі у необхідні місця. Однією з головних особливостей є велика кількість різного роду комбінацій клавіш, які, при достатніх навичках роботи, дозволяють повноцінно писати програми майже або зовсім не користуючись мишкою. В них є ще багато іншого функціоналу, який сильно спрощує роботу різного роду.

Іншим прикладом програм для роботи з різного роду текстами, такими як документи або літературними виданнями, є текстові редактори, такі як браузерний редактор тексту Google Docs, безкоштовний додаток Wordpad, або потужний редактор Word, усі вони дозволяють перевіряти коректність грамматики введеного тексту в залежності від мови, на якій написаний текст, мають окремі панелі для налаштування формату тексту, його колір, стиль, додавати зображення різних форматів, нумеровані списки, додавати нумерацію для них, налаштовувати усі види відступів, тобто усе що може знадобитися для створення документів або текстів будь-яких форматів.

Вже існує дуже багато різного роду текстових редакторів з великим обсягом функціоналу, проте тема гідроакустичних експериментів є дуже вузькою, під яку немає налаштованих серед розробки, так як розроблена мова є унікальною. Через це і була поставлена задача створення окремого редактора тексту для опису гідроакустичних експериментів.

Мотивацією для створення генератора тексту слугує об’єм тексту опису, через що від науковця вимагалося б пам’ятати повний набір усіх об’єктів та їх властивостей, крім того ще й пам’ятати порядок слідування як об’єктів, так і властивостей, що зводить вірогідність помилки до дуже високої.

1. ЗАДАЧА СТВОРЕННЯ ЛІНГВІСТИЧНОГО ЗАБЕЗПЕЧЕННЯ ОПИСУ СЦЕНИ ПРОВЕДЕННЯ ГІДРОАКУСТИЧНИХ ЕКСПЕРИМЕНТІВ

Метою представленої дипломної роботи є створення, на основі іншої системи, яка займається проведенням гідроакустичних експериментів, системи для опису цих експериментів, тобто інструментом, який може керувати початковими умовами експериментів. Створений програмний продукт має вирішувати такі задачі:

розробити мову для опису експерименту;

реалізувати транслятор для розробленої мови;

створити середу для зручної роботи з описом експерименту;

автоматизувати друк шаблонного тексту;

1.1 Задача розробки мови для опису експерименту

За своєю структурою, експеримент має декілька рівнів вкладеності. об’єкт експерименту складається з полів назви та опису, а також включає в себе об’єкти сцени та сценарія. Сцена у свою чергу складається з налаштувань сцена та морського об’єкта, він має назву та технічні характеристики. Сценарій же складається з опису характеру руху об’єкта під час експерименту. Через усі ці необхідності мова опису має бути розділена на 4 логічні частини, при цьому зберігаючи можливість розрізняти кожну з них і не давати користувачу плутатись між ними. Розроблена мова має чітко та інтуїтивно описувати експеримент, мінімізуючи кількість усіх службових символів. Бути легкою для орієнтування у

тексті, для пошуку необхідного поля необхідного об’єкту. Також мова має бути простою для сприйняття науковцем, який вперше бачить текст опису на ній і не потребувати знань програмування, лише знання структури гідроакустичних експериментів.

1.2 Задача реалізації транслятора для мови опису експерименту

Для використання будь-якої розробленої комп'ютерної мови необхідно мати транслятор. Він виконує роль перекладача тексту, написаного мовою опису на потрібний вихідний формат. Необхідно вибрати зручний вихідний формат для транслятора, який буде зручний для системи проведення гідроакустичних експериментів. Транслятор надавати можливість легко розширювати структуру мови, додаючи нові поля та об’єкти. Не займати занадто багато часу на трансляцію, давати інформативні повідомлення про помилки у описі експерименту, які дозволять достатньо швидко науковцю знайти місце помилки та виправити його.

1.3 Задача створення середи для зручної роботи з описом експерименту

Користувачі зазвичай не хочуть працювати з складними текстами з допомогою лише звичайного текстового редактора, бо тоді їм доведеться виконувати велику кількість роботи, наприклад контролювати правильність написаного тексту, власноруч зберігати, зчитувати та транслювати файли опису. Створена середа має позбавляти користувача від усіх цих обов'язків, потребуючи від нього пам’ятати

лише значення необхідних полів. Також має бути візуальне розділення між структурними елементами опису, можливість відкривати, створювати та транслювати файли опису.

1.4 Задача автоматизації друку шаблонного тексту

Для позбавлення користувача необхідності друкувати однотипний текст, так як більшу частину описи складають ключові слова, які по суті не змінюються між експериментами, необхідно розробити генератор опису. Він має представляти собою форму для введення значень полів, до кожного об’єкту окрема форма, мати можливість зручно переходити між формами, та фактично дублювати структуру мови опису візуально. Форма має включати в себе валідацію основних полів, для позбавлення користувачів можливості випадково помилитися у форматі, а також потребувати введення хоча б мінімально необхідної кількості полів, які необхідні для створення опису.

2. ЛІНГВІСТИЧНЕ ЗАБЕЗПЕЧЕННЯ ОПИСУ СЦЕНИ ПРОВЕДЕННЯ ГІДРОАКУСТИЧНИХ ЕКСПЕРИМЕНТІВ

Лінгвістичне забезпечення є основоположними у сучасному інформаційному світі. Існує велика кількість різних інформаційних систем з дуже складною структурою, опис яких на машинній мові зайняв би дуже велику кількість часу та зусиль. Через це було створено велику кількість різних мов програмування, які зводять велику кількість машинних команд до однієї, більш високорівневої.

Найбільш розповсюдженими мовами програмування є мови загального призначення, які можуть описувати логіку систем з будь-яким рівнем складності та призначенням. Фактично усі мови загального призначення мають все необхідне для опису сцени проведення експериментів з гідроакустики, інколи ці мови можуть описати розроблену мову меншою та більш інформативною кількістю символів, але потребували б від науковців деякої підготовки.

Усі мови програмування мають деякі спільні риси, так як усі вони виконуються на електро-пристроях, які мають однакову фундаментальну структуру. Усі вони мають процесор, спосіб зберігати дані, якісь способи зв’язку між ними. Крім того спосіб зберігання інформації базується на одних і тих самих речах.

Ці фактори вплинули на структуру перших мов програмування, таких як C та C++, ці мови славляться своєю оптимальністю роботи, яка доволі чітко описує як само програма звертається до пам’яті пристрою. А більш сучасні мови, як відомо, основують свою структуру на більш ранніх мовах, через це усі сучасні мови і мають спільні риси, так як мають спільний фундамент.

По цим причинам науковцю, щоб користуватися сучасними високорівневими мовами загального призначення необхідно розуміти цей мовний базис, що є недопустимим, так як потребує дуже великої кількості часу та зусиль.

Якщо проігнорувати від науковця необхідність вивчення мови повністю, все одно синтаксис обраної мови буде потребувати введення деяких службових символів, які слугують лише описом структур, необхідних для функціонування самої мови, що ускладнює сприйняття опису сцени навіть для досвідченого користувача.

2.1 Опис структур даних за допомогою мов програмування загального призначення

У якості демонстрації можно взяти опис користувача за допомогою мови загального призначення Java. Ця мова є об’єктно-орієнтованою, тобто увесь процес роботи програм, написаних на цій мови фактично представляє собою взаємодію окремих об’єктів, кожен об’єкт представляє собою згрупований набор змінних та методи для роботи з ними. Структура об’єктів описується за допомогою класів, існує доволі гарна аналогія, простіше зрозуміти клас як креслення будинку, а об’єкт класу як збудований будинок, класс завжди один а об’єктів цього класу може бути багато. Опис структури користувача за допомогою класса цієї мови є на Рисунку 2.1.

```
public class User{
  private String email;
  private String password;
}
```

Рисунок 2.1 — Опис класу користувача з полями емайла та пароля на мові програмування Java

На цьому описі можно побачити доволі велику кількість ключових слів та символів які слугують лише описами структур самої мови, наприклад ключові слова private та public це модифікатори видимості, які дозволяють чи забороняють іншим классам використовувати класс або змінну на якій стоїть цей модифікатор, також

можна побачити ключове слово клас, яке інтуїтивно ніяк не відноситься до користувача, а також типи даних полів класу, у цьому випадку String, тобто строка. Напряму до структури користувача відноситься лише ім’я класу User та імена полів email та password. Також тут присутні фігурні дужки, які позначають початок та кінець тіла класу та крапки з комою, які говорять компілятору про те що команда закінчилась. Приклад гарно описує чому більшість загальних мов програмування є багатослівними та не підходять для реалізації опису, проте існують і інші мови загального призначення.

Інший приклад написаний за допомогою високорівневої мови для розробки Web-додатків JavaScript. Ця мова програмування JavaScript э нестрого типізованою та динамічною, з самого початку вона створювалася як спрощений формат мови Java, який використовувався би браузерами для додання складної логіки веб-сторінкам, проте з часом почала використовуватися у багатьох напрямках. Її спосіб типізації дозволяє користувачу не думати про типи даних та сконцентруватися на логіці роботи програми, що часом призводить до помилок типів, проте більш протим синтаксисом. На Рисунку 2.2. можна побачити опис аналогічної, до минулого прикладу, структури користувача.

```
const user = {
  email: 'user@gmail.com',
  password: '12345'
}
```

Рисунок 2.2 — Структура користувача з полями емайла та пароля на мові програмування JavaScript

На цьому рисунку видно що ця мова потребує меншої кількості ключових слів і навіть дозволяє відразу вказувати значення полів, що є набагато більш лаконічним за минулий приклад. У прикладі є лише одне ключове слово const, яке об’являє змінну та забороняє її змінювати. Але службових символів стало більше, наприклад одинарні лапки вказують на те що між ними знаходиться строка, також символ

рівності записує створену структуру у змінну, фігурні дужки позначають створення об’єкту, а двокрапки вказують значення полям. Хоча такий синтаксис і є більш простим с точки зору програмування, для розуміння науковцями він усе ще складний.

Як відомо, вже створено багато мов програмування, і з кожним роком з’являються нові, які дозволяють робити щось нове, чого не було у мовах створених раніше. Деякі мови дозволяють за допомогою своїх власних синтаксичних конструкцій описувати текстом, близьким до звичного більшості людей, фактично створювати у середині себе предметно-орієнтовані мови.

Kotlin є відмінним прикладом мови, на основі якої можливо будувати предметно-орієнтовані мови. Kotlin це мови загального призначення, яка увібрала в себе особливості багатьох інших мов. Спочатку ця мова проектувалася лише під JVM, що надає їй можливістю бути повністю сумісною с Java кодом, проте маючи багато переваг. Ця мова може суміщати як об’єктно-орієнтований підхід, так і функціональний, відкриваючи багато можливостей для побудови різних синтаксичних реалізації однієї і тієї самої логіки. Приклад опису структури користувача з минулих прикладів є на Рисунку 2.3.

```
User createWithEmail "user@gmail.com" andPassword "12345"
```

Рисунок 2.3 — Структура користувача описана за допомогою предметно-орієнтованої мови, реалізованої на Kotlin

Отриманий у результаті текст є дуже близьким до звичайного мовного опису цієї структури. Фактично у прикладі нема ключових слів самої мови, а з символів присутні подвійні лапки, для позначення того що введений текст є строковим значенням. Це означає що за необхідності можливо замінити всі назви за необхідністю. Фактично приклад складається з двох функцій та класу. Коли функція у мові Kotlin має два параметри то її можливо переписати у інфіксному вигляді,

тобто розмістити параметри з боків, а посередині написати ім’я функції і компілятор переведе це у необхідний формат.

Слова createWithEmail та andPassword є інфіксними функціями, перша на вхід приймає об’єкт користувача та додає до нього значення емейлу та повертає об’єкт користувача, після чого друга функція додає до нього значення пароля та повертає його. Хоч такий спосіб значно кращий за минулі варіанти, він має декілька фатальних недоліків. Найбільш очевидним є необхідність відповідати правилам найменування мови, тому наприклад не можна дати ім’я функції з пробілом, через що доводиться зливати всі слова разом та кожне наступне слово писати з великої літери, також стиль цієї мови потребує писати всі назви функцій за допомогою англійських літер, проте основним недоліком є складність переносів, через що дуже довгі та великі структури доведеться писати в одну строку, що стає неможливо читати, чи не інтуїтивно переносити, що сильно відобразиться на сприйнятті і також заважає необхідність писати подвійні лапки для строкових змінних.

2.2 Опис структур даних за допомогою предметно-орієнтованих мов

Наведені приклади демонструють багато різних недоліків більшості універсальних мов, проте існують і інші види мов, так звані предметно-орієнтовані мови, вони беруть основні синтаксичні елементи своєї предметної області та будують на їх основі мову, видаливши всі службові символи які не відносяться до того що описується цією мовою, фактично у рамках самої предметної області описуючи її структури.

Відмінним прикладом є мова MySQL. Вони використовується для створення запитів до баз даних, які дозволяють працювати з таблицями. Ці запити можуть

створювати або видаляти таблиці, змінювати їх структуру, зв’язувати таблиці та отримувати дані з таблиць, групуючи їх за необхідності, що дозволяє гарантувати цілісність збереженої інформації. Як приклад можно взяти запити на створення таблиці користувача з попередніх прикладів, додавання до неї нового запису та отримання усіх записів з таблиці, що зображено на рисунку 2.4.

```
CREATE TABLE User(email varchar(100), password varchar(100));
INSERT into User(email, password) values("user@gmail.com", "12345");
SELECT * FROM User;
```

Рисунок 2.4 — Запити для створення таблиці користувача, додавання в неї нового запису та отримання усіх записів на мові запитів MySQL

З першого погляду може здатися що синтаксис запитів має безліч лише функціональних ключових слів та службових символів, проте всі вони напряму описують елементи предметної області. У прикладі є доволі багато ключових слів, проте всі вони чітко описують що саме робить кожен окремий запит, так перший явно створює таблицю, другий додає дані, а третій вибирає дані з таблиці. Також присутнє слово varchar, з першого погляду це схоже на аналог String, проте у цьому випадку це описання типу поля у середині таблиці, а так як таблиця є частиною бази даних то це ключове слово описує саме внутрішній елемент предметної області і тому не є службовим. Останнє ключове слово FROM несе більш спрощуючу для сприйняття роль та слугує візуальним роздільником між сусідніми символами, так як поряд їх було б доволі складно сприймати. Це ключове слово слугує більш ролі спрощення сприйняття для користувача, проте це також не робить його службовим, аналогічну роль мають фігурні дужки та коми, відділяючи окремі смислові елементи на фоні інших. Символ зірки є скороченням, яке позначає що при зчитуванні даних з таблиці необхідно взяти усі записи.

MySQL є дуже гарним і інформативним прикладом створення мови під певну предметну область, дозволяючи користувачу провести інтуїтивні ассоціації з об’єктами які описуються мовою. Єдиним недоліком цієї мови є службовий символ

закінчення команди, тобто крапка з комою. Вона необхідна лише для спрощення роботи транслятора, так як ця мова є достатньо старою, коли швидкість була доволі важливою.

Сучасний розвиток технологій робить такого роду оптимізації непотрібними у більшості випадків. Тому вони вже не є необхідністю для трансляторів, і залишаються у мовах які створені доволі давно, таких як JavaScript, Java, C++, Pascal, та у тих які концентруються на оптимізації роботи, так як використовуються для створення драйверів чи інших фундаментальних речей, до них відноситься C, Rust. Новіші мови беруть за основу на спрощення синтаксису, сприйняття та збільшення інтуїтивності написаного коду, що значно прискорює роботу розробника з вже написаним кодом, за цей рахунок трохи уповільнюється робота самої мови програмування, до них можна віднести Kotlin, Swift, Go. Але цей підхід мав користувачів і раніше, доволі гарною демонстрацією цього є мова Python, яка була націлена на спрощення основного синтаксису для розділення коду на набір команд, окрів крапок з комою, прибираючи і фігурні дужки, які слугували описом для тіла методу або класу, націлюючи описання всіх блоків коду на пробіли, на перший погляд це запрошує сприйняття, проте з іншої сторони це ускладнило написання коду для розробників, бо кожен зайвий пробіл має можливість зламати роботу коду.

2.3 Мова опису експерименту

Розроблена мова опису експериментів бере до уваги усі ці недоліки, як мов загально-орієнтованих так і предметно-орієнтованих, та ставить у якості цілі спрощення роботи з мовою опису, базуючись на практичності роботи, проте враховуючи і швидкість введення нового тексту чи його читання користувачем. Через це розроблена мова, аналогічно до Python використовує символи переносу для

розділу різних команд, що робить для користувача цю операцію непомітною, а також заважає користувачу описувати усі властивості об’єктів одним рядком, що значно ускладнює б сприйняття. Також присутня можливість проставляти пробіли перед початками строк для більш просто сприйняття вкладеності, проте на відміну від мови програмування Python, це не зіпсує. Такка структура мови направляє користувача до написання в певному оптимальному, для сприйняття при читанні, стилі. Це продемонстровано на Рисунку 2.5.

```
експеримент
    назва Назва експерименту
    опис Опис експеримента
    сцена
        назва Назва сцени
        профіль солености (0.0;0.0) (0.0;0.0) (0.0;0.0)
        профіль температури (0.0;0.0) (0.0;0.0) (0.0;0.0)
        центр (0.0;0.0)
        час моделювання 0
        глибина 0
        морський об’єкт
            назва Назва морського об’єкта
            ТТХ [0.0] [0.0;0.0] [0.0;0.0;0.0]
    сценарій
        назва об’єкту Назва морського об’єкта
        початок руху (0.0;0.0)
        напрям 0
        швидкість 0
        глибина 0
        шлях до моделі C:\Users\user\AppData\Roaming\Microsoft
```

Рисунок 2.5 — Опис гідроакустичного експерименту но розробленій мові опису

У мові опису не має службових символів, усі синтаксичні конструкції слугують лише забезпеченню інтуїтивності сприйняття написаного тексту. Розуміння описаної структури легко дається людині яка вперше бачить цю мову, проте володіє знаннями самої предметної області. Опис складається з ключових слів та їх значень. Ключові слова у свою чергу підрозділяються на назви об’єктів та їх поля. Усі ключові слова знаходяться на початку рядка, проте можуть мати перед собою-будь яку кількість пробільних символів, що дозволяє користувачу більш інтуїтивно описувати вкладеності об’єктів. На відміну від назви об’єкту, після назви

поля завжди має слідувати певне значення. Значення бувають трьох типів: строка, двовимірна точка або група. Строкою може бути будь-який символів, дозволяється навіть вставляти у них ключові слова, транслятор легко відрізняє їх від ключових слів. Точка представляє собою математичну двовимірну точку і позначається відповідно за допомогою дужок та крапки з комою. Групу можно уявляти як багатовимірну точку, проте вона несе зовсім інший синтаксичний сенс, Гурппа може складатися з будь-якої кількості координат, аналогічно до точки між ними мають стояти крапки з комою, а замість круглих дужок використовуються квадратні, для спрощення сприйняття відмінності точок від груп. Групи та точки у своїх полях можуть вказуватись у будь якій кількості, тобто фактично поля, які використовують точки та групи, завжди можуть мати список точок чи груп.

2.4 Аналогічні системи

На сьогоднішній день є не так багато мов які описують структуру гідроакустичних експериментів, але є схожі системи, у якості прикладу може виступати система створена у результаті виконання дипломної роботи по темі: “Лінгвістичне забезпечення сценаріїв і сцен моделюючих комплексів” [1]. Метою цієї роботи було створення мови, а також графічного інтерфейсу для інтерпретації сценаріїв обробки інформації у модулюючих комплексах.

У результаті була створена якісна мова опису, яка нагадує мову опису експерименту, проте її основною ціллю є опис сценаріїв. У мові опису сценаріїв і сцен використовувалися символи крапок з комою та характерні, для найменування змінних мови Python, нижні підкреслення, це було необхідно через деякі проблеми розбору транслятором складних послідовностей слів з великою кількістю пробілів, такого роду заходи насправді спрощують трансляцію тексту та прискорюють роботу

програми. Ця програма є доволі гарним прикладом націлення на швидкість трансляції, що дає їй деяку перевагу у порівнянні зі створеною системою опису експерименту, проте розроблена мова опису більш проста у сприйнятті а також написанні.

2.5 Структура гідроакустичного експерименту

Гідроакустичний експеримент має складну структуру з декількома рівнями вкладеності. На найвищому рівні знаходиться об’єкт самого експерименту, всередині якого знаходяться всі інші. Експеримент має назву та опис, які можуть виступати будь-якими реченнями з необмеженими наборами символів. Експеримент включає в себе об’єкти сцени та сценарію.

Сцена представляє собою середу, у якій проводиться експеримент. Аналогічно до експерименту, сцена має свою власну назву, а також профілі солоності та температури, вони описують параметри води, які впливають на характер руху гідроакустичних сигналів, та представляють собою список точок. Центр представляє собою двовимірну точку, яка є координатами акваторії, тобто місця з якого починаються усі обрахунки а також слугує місцем я кому обраховуються результати експерименту. Час моделювання описує кількість секунд яку буде проходи експеримент. Глибина це відстань від поверхні води до дна. Сцена також має морський об’єкт.

Морський об’єкт представляє собою рухому сутність яка і породжує гідроакустичні сигнали. Він має назву, яка потім використовується у сценарії, а також технічні характеристики, які представляють собою комплексний збір усіх можливих характеристик, які ще досліджуються і не є повністю визначеними.

Сценарій слугує елементом для опису характеру руху об’єкту, у полі імені об’єкту завжди має бути існуючий морський об’єкт. Поле початку руху це місцеположення морського об’єкту до початку експерименту та місце з якого він починає свій рух. Поле напряму позначає кут під яким буде рухатись об’єкт під час експерименту. Швидкість позначає відношення середньої кількості пройденої відстані до одиниці часу та використовується для обчислення положення об’єкту у будь-який момент часу. Глибина позначає відстань від поверхні води до об’єкта, у разі якщо він рухається під водою. Шлях до моделі це шлях до файлу з моделлю яка використовується для обрахунків сценарію.

2.6 Висновки до розділу

Існує дуже велика кількість різних мов для опису, проте усі вони мають свої слабкі та сильні сторони, тому під час створення своєї мови необхідно підстроювати їх під потреби поставлених задач.

Так проблемою мов загального призначення є велика кількість службових символів. Хоч часто ці символи і дозволяють прискорити трансляцію, у сучасних реалях такого роду оптимізації не дають значного виграшу у більшості випадків, що є актуальним і у випадку розробленої системи. Проте інколи ці мови можуть мати особливості синтаксису, які дозволяють сильно наблизитись до інтуїтивного, але усе ж таки маючи свої обмеження.

Предметно орієнтовані мови слугують гарною основою для розуміння того якою саме має бути мова, орієнтована на створення опису сценаріїв. Головним недоліком предметно-орієнтованих мов є вузькість області їх застосування. Орієнтування на розуміння написаного тексту людиною яка знає предметну область,

але вперше працює з мовою є доволі гарною практикою, бо мінімізації порогу входу є дуже важливою.

Розроблена мова увібрала в себе головні особливості інших мов, проте позбавилась від їх недоліків, щоб зробити її оптимальною для опису експерименту. Мови експерименту не має службових символів, усі слова та символи мають пряме відношення до гідроакустичних експериментів та повністю дублюють їх структуру.

Структура мови адаптована для опису структур з багатьма рівнями вкладеності за допомогою пробілів, а також може читатися як звичайний текст науковцем, який вперше бачить текст опису, а отже система повністю відповідає усім необхідним умовам. Також для позначення строкових змінних не потрібно використовувати лапки, так як транслятор завжди може чітко розрізнити ключове слово та значення поля.

3. ЗАСОБИ РОЗРОБКИ

Програмний продукт був реалізований на мові програмування Kotlin, яка базується на JVM, для створення інтерфейсу була використана бібліотека TornadoFx, в основі якої лежить популярний Java фреймворк JavaFx. У якості системи зборки залежностей використовується Maven, а для розробки була обрана середа IntelliJ IDEA, розробники якої також є розробниками основної мови реалізації програмного продукту, що вносить доволі велику підтримку при розробці з боку середи, яка є однією з найкращих при роботі з Java-подібним кодом.

3.1 Мова розробки

Основною мовою розробки був обраний Kotlin. Ця мова є строго типізованою, об’єктно-орієнтованою і працює на основі JVM. Вона створена у JetBrains, ця компанія в основному займається створенням середовищ для програмування, зокрема їми була створена IntelliJ IDEA. Kotlin була створена так, щоб бути повністю сумісною з Java, беручи за основу її основні принципи та удосконалюючи їх. Це дає великий вибір способів реалізації логіки під час написання коду.

3.1.1 Мова Java

Java це статично типізована об’єктно-орієнтована мова програмування, розроблена у компанії Sun Microsystems, проте зараз вона належить компанії Oracle, перша версія мови була представлена пуу 1995-му році. На даний момент на мові програмування Kotlin є актуальною версією мови, ця версія була презентована у квітні 2020-го року. Зараз Java є однією з найбільш популярних мов програмування, велика кількість компаній використовує її для написання, як клієнтської[2], так і серверної частини своїх додатків. Вже створено багато десктопних та мобільних додатків, написаних на цій мові, і багато з них користуються доволі великою популярністю. усі додатки системи Android були написані на Java, чи скомпліковані у її байт-код. В основі структури Java знаходяться об’єкти, з яких будуються усі властивості та взаємодії, хоча це інколи обмежує розробників, у більшості випадків це вирішує велику кількість архітектурних проблем, так як об’єктно-орієнтований стиль більш просто описує код ніж навіть функціональний стиль програмування. Проте головною причиною популярності саме цієї мови програмування вважається JVM.

Компіляція Java коду у машинні коди відбувається у декілька етапів. Спочатку код, написаний на цій мові, перетворюється у так званий проміжний байт-код, після цього отриманий код передається у JVM, яка перетворює його вже у машинні коди потрібного пристрою. Ця схема проілюстрована на рисунку 3.1.

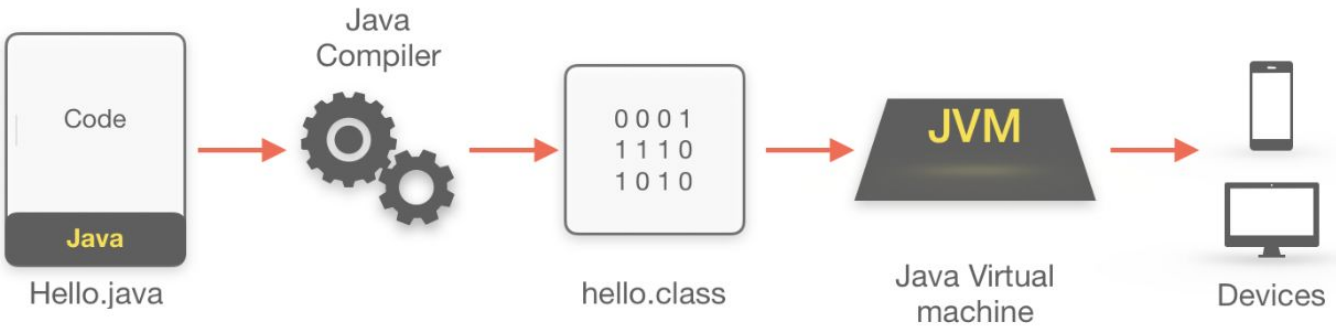

Рисунок 3.1 — Принцип роботи компілятора мови Java та JVM

Головною причиною такої кількості кроків є кросс-платформеність. Вона полягає у тому що написаний на Java код можна скомпілювати у код для будь якої системи чи пристрою, усе це відбувається за допомогою саме JVM. Віртуальна машина розуміє лише байт-код і може перевести його у необхідний машинний код, через це спочатку джава компілятор переводить Java код у формат зрозумілий для віртуальної машини. Саме це і дозволяє коду цієї мови запускатися як на Windows так і на Android або будь-якій іншій системі. Через цю свою особливість Java спочатку позиціонувалася як мова для розробки пультів та кавомашин. Іншим дуже великим плюсом саме Java є автоматичний збірник сміття, який контролює усю роботу з пам'яттю та робить неможливими втрати пам'яті. На тих же принципах базується і мова Kotlin, яка стала подальшим ідейним розвитком Java.

3.1.2 Мова Kotlin

Як було сказано раніше, мова Kotlin була створена у якості ідейного продовження Java. Вона бере гарні сторони свого попередника, такі як JVM та збірник сміття, і додає свої, більш прості та практичні синтаксичні конструкції, можливість компіляції напряму у код платформи, або навіть компіляцію у мову JavaScript, яка активно використовується для веб-розробки. Розробка цієї мови почалася у 2010-му і на початку 2016-го була опублікована перша версія мови. Kotlin увійшов в топ мов, як мова з найбільшою швидкістю росту аудиторії. По більшій частині ця мова отримала свою популярність саме через повну сумісність з Java, на стільки що в одному проекті може бути одночасно код обох мов і все буде добре компілюватися і працювати. У середині 2017-го року Kotlin став основною мовою розробки під Android, що дало впевненість у популярності мови на довгий час.

В основі Kotlin лежить прагматичність, ця мова створювалась з ціллю бути зручною у розробці, суміщаючи простоту, читаємість та зменшення кількості помилок. Також розробники використовували досвід інших мов і переняли конструкції з таких мов як Pascal, TypeScript, Go, C#, Scala, Rust, Python і багато інших. Проте, замість хаосу який міг би утворитися від такої кількості різних підходів, ця мова групує особливості інших мов у свою власну ідеоматику, роблячи свою структуру цілісною.

У порівнянні с Java, Kotlin має багато своїх особливостей. По-перше це підтримка функціонального програмування на рівні мови [3], що дозволяє суміщати об’єктно-орієнтований та функціональний підхід у програмуванні. Іншим дуже важливим підходом є null-безпека, що розділяє усі змінні на ті у яких може бути null та на ті у яких він бути не може, забороняючи виклик функцій у яких може бути null. Це дозволяє уникнути дуже великої кількості помилок, які трапляються у Java, зокрема так званой “Помилки на мільйон”, яка сталася саме через null. Ще однією дуже приємною особливістю є можливість описувати так звані POJO клас, який використовується у якості елемента предметної області і є його представленням у коді, як наприклад, класс точки, який має лише дві координати, у Java для об’явлення цього классу було б потрібно об’явити два поля, конструктор, методи отримання та присвоєння значень цим полям, перегрузити методи порівняння та метод переводу у строковий вид, що може зайняти до 30 строк коду, у той час як на Kotlin це займає лише один рядок. Приклад класу “точка” можна побачити на рисунку 3.2.

```
data class Point(var x: Int, var y: Int)
```

Рисунок 3.2 — Реалізація POJO класу “точка” на мові Kotlin

Така мініатюрність класу реалізується за рахунок того що POJO класс є шаблонним для мови Java, і під час його опису розробникам доводилось написати багато шаблонного код для кожного такого класу. Компілятор Kotlin бере на себе

генерацію такого шаблонного коду, спрощуючи опис до одного ключового слова data. Ця мова у багатьох випадках дозволяє сильно скоротити об’єм коду за рахунок позбавлення від шаблонного коду, дозволяючи розробнику концентруватися не на написанні коду, а на його обдумуванні. Також дуже важливими є корутини. Вони дозволяють написати багатопоточний код, не утворюючи при цьому окремих потоків, так як операція створення нового потоку є доволі затратною. Корутини це так звані легковісні потоки, які можуть існувати у рамках одного потоку, не блокуючи його, так, наприклад, коли одна корутина чекає даних від іншої, третя може спокійно виконуватися у цьому ж потоці. Також корутини балансують між потоками, оптимізуючі свою роботу. Можна створювати коритини тисячами, ніяк не сповільнюючи роботу системи, у той час як створення і сотні потоків витратить доволі багато ресурсів. Аналогічно до інших синтаксичних структур, корутіни можуть бути написані у різних стилях, починаючи від async та await як у JavaScript, та закінчуючи стилем promise.

Наостанок розглянемо побудову предметно-орієнтованих мов за допомогою Kotlin. Однією з найпомітніших особливостей мови Kotlin є зручність побудови DSL за допомогою її синтаксичних конструкцій. Для цього використовуються так звані інікснi функції, які фактично перетворюють виклик функції з двома параметрами у щось дуже схоже на виклик оператора. Порівняння можно побачити на рисунку 3.3.

```
"123".append("4")

"123" append "4"
```

Рисунок 3.3 — Порівняння виклику звичайної функції та інфіксного аналога

Фактично перетворення інфіксної функції у звичайну це позбавлення розробника необхідності писати символи крапки та дужок, що часто є дуже зручним при побудові предметно-орієнтованих мов. Також важливо згадати спрощення лямбд, що дозволяє не писати круглі лапки коли лямбда це останній параметр фунції. Приклад можна побачити на рисунку 3.4.

```
listOf(1, 2, 3, 4).filter({ it % 2 == 0 })

listOf(1, 2, 3, 4).filter{ it % 2 == 0 }
```

Рисунок 3.4 — Приклад спрощення лямбд у мові Kotlin

У мові Kotlin лямбди визначаються за допомогою фігурних дужок і якщо вона має один єдиний параметр то до нього можна звернутися за допомогою ключового слова it. Для прикладу була взята функція фільтрації списку, фактично лямбда повертає true коли число парне та false коли воно непарне. Функція filter повертає список лише з тими елементами, на які лямбда повернула істину, тобто лише на парні. Фактично вирази зверху та знизу є еквівалентними, проте вираз знизу є більш простим для написання та сприйняття. Використовуючи інфіксні функції та лямбди можна побудувати предметно-орієнтовану мову як на рисунку 3.5.

```
@Test
fun `when 'or' conditions are specified then they are respected`() {
    doTest("select * from table1 where (column3 = 4 or column4 is null)") {
        from ("table1")
        where {
            or {
                "column3" eq 4
                "column4" eq null
            }
        }
    }
}
```

Рисунок 3.5 — Приклад DSL на мові Kotlin

На рисунку зображений приклад тесту запитів за допомогою DSL мови Kotlin. У першому рядку знаходиться анотація яка говорить тестовому фреймворку про те що функція нижче є тестом. У другому рядку знаходиться об’явлення функції, за допомогою ключового слова fun, між специфічними лапками, які є ще однією корисною особливістю мови Kotlin, знаходиться назва функції, такі лапки дозволяють давати назви будь-яким классам, змінним та функціям з символами

розділення, такими як пробіл, кома, інші лапки. Такі імена є доречними у випадках тестових функцій. У наступних рядках викликаються функції, які на вхід приймають лямбди, у середині яких викликаються інші функції, таким чином реалізується вкладеність у DSL та наближення коду до інтуїтивного рівня просто тексту, який може читати і повністю розуміти будь-який спеціаліст у певній предметній області.

Саме через таку оптимальність роботи з мовою, яку дають її особливості вона і була обрана у якості основної. Головною особливістю Kotlin є її строга типізація, ідеоматичність та строгість у конструкціях, це не дозволяє коду перетворитися на хаос, так як сама мова підтримує якісний стиль кодування. Основною популярності вона набула серед Java розробників, так як Kotlin базується на дуже добре знайомій їм, спрощуючи великі і складні конструкції.

3.2 Фреймворк TornadoFx

Для реалізації графічного інтерфейсу був обраний фреймворк TornadoFx. Це фреймворк, який базується на JavaFx, проте удосконалює його за допомогою конструкцій мови Kotlin, будуючи зручну предметно-орієнтовану мову для опису розмітки об’єктів у формі та їх стилів, які базуються на форматі CSS.

За допомогою цього фреймворку можливо зручно створювати форми для введення даних, додавати валідацію для полів. Приклад такої форми можна побачити на рисунку 3.6.

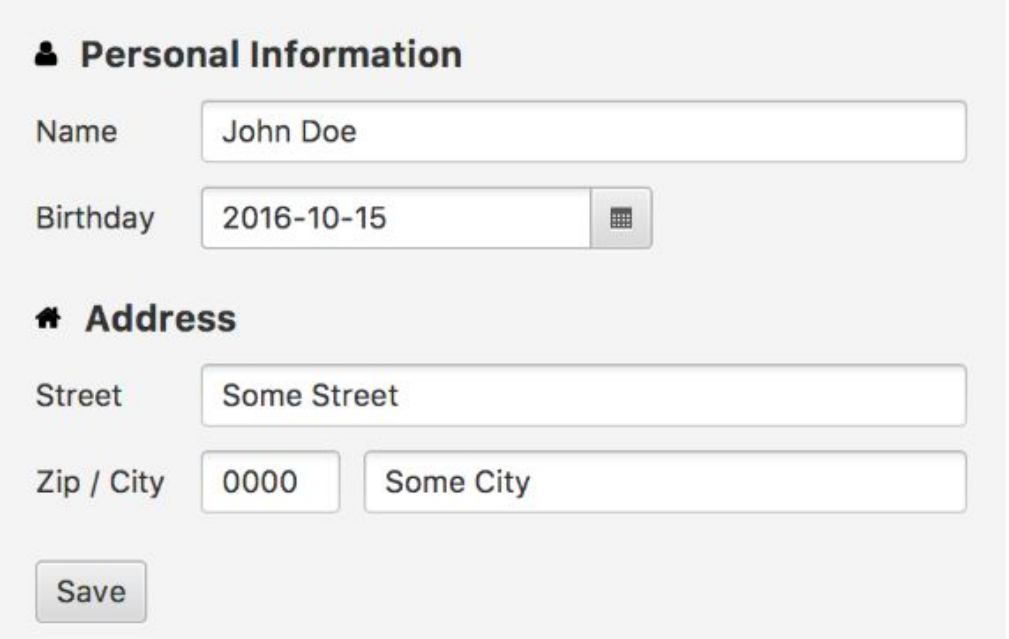

Рисунок 3.6 — Приклад форми, реалізованої за допомогою TornadoFx

Представлена форма слугує для отпримання інформації про людину, такой як ім’я, дата народження, адресса. Для реалізації використовувались звичайні текстові поля, та окреме поле для вводу дати, Схожі форми доволі легко створювати за допомогою цього фреймворку, саме через цю простоту він і був обраний.

4.3 Висновки до розділу

Всі обрані для реалізації інструменти повністю охопили весь необхідний функціонал та забезпечили якісне виконання поставлених задач. Основний код був написаний на мові програмування Kotlin, яка дозволила у більш простій та зрозумілій формі написати програму транслятора, а також, за допомогою бібліотеки TornadoFx, у вигляді предметно-орієнтованої мови, яка була реалізована за допомогою Kotlin, описати графічні вікна, з якими відбувається робота. Жодних проблем у роботі бібліотеки чи мови не було виявлено.

4. ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ

Реалізована система по своїй суті є транслятором із зручним графічним інтерфейсом. Графічний інтерфейс розділений на генератор тексту та редактор тексту. Структурно програмна реалізація має 4 ключові частини, усі вони зображені на рисунку 4.1.

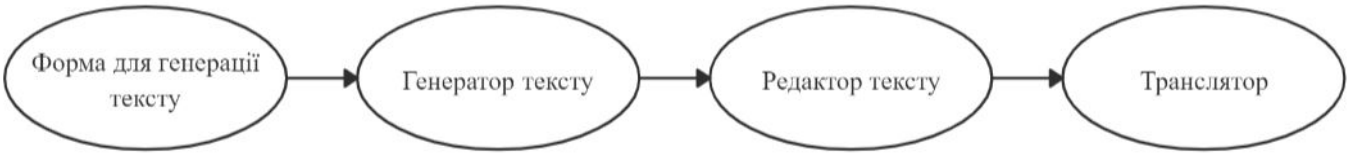

Рисунок 4.1 — Опис основної структури системи

4.1 Графічна частина реалізації

Форма для генерації тексту це форма, яка складається з декількох підрозділів, кожен з яких представляє окремий об’єкт експерименту, це було створено з ціллю спрощення заповнення вхідних даних, так як всі поля разом займають доволі багато місця. Структура розділів повторює вкладеності самих об’єктів цих розділів. Через те що кожна сцена може мати багато об’єктів, а кожен експеримент включати декілька сценаріїв існує можливість зручно створювати нові форми сцен та сценаріїв, зберігаючи візуальну структуру вкладеності. На виході форма генерації тексту повертає об’єкти предметної області з описаними у формі значеннями полів та описаною структурою вкладеності.

Генератор тексту у свою чергу трансформує об’єкти експерименту у текстову форму, зберігаючи все необхідні для мови опису правила, генеруючи назви об’єктів, властивостей, подвійні лапки для позначення строкових змінних та точки, згідно

формату мови, фактично створюючи повністю робочий опис експерименту. Також генератор тексту зберігає відступи для збереження візуальної вкладеності, позбавляючи користувача від необхідності розставляти пробільні символи саморуч.

Редактор тексту представляє собою середу для зручної роботи з текстом, підсічуючи необхідні ключові слова, дозволяючи редагувати додавати чи видаляти дані з мови. Фактично редактор тексту може працювати незалежно від форми для генерації тексту та генератора тексту, але тоді користувачу доведеться власноруч вводити увесь опис експерименту, що дуже легко може призвести до помилок. Після завершення вводу, текст потрапляє до транслятора, який переводить його у JSON формат.

4.2 Реалізація транслятора

Останній та найскладніший структурний елемент системи це транслятор. Транслятор приймає на вхід текст, написаний на мові опису експерименту, перевіряє його на валідність та конвертує його у необхідний системі проведення експериментів формат. Трансляція вхідного тексту відбувається у три етапи [4], кожен з яких можна побачити на рисунку 4.2.

Рисунок 4.2 — Схема трансляції вхідного тексту у вихідний

4.2.1 Лексичний аналіз

Першим етапом трансляції є лексичний аналіз, який виконується за рахунок лексичного аналізатора, перетворюючого вхідний текст у таблицю лексем [5]. Спочатку аналізатор розділяє слова на окремі лексеми, за допомогою пробілів та деяких інших ключових символів, далі він перевіряє кожну лексему на коректність, виставляючи відповідний тип кожній лексемі, якщо у системі немає типу для отриманої лексеми видається повідомлення про помилку, у якому вказана сама лексема, текст про те що лексема не розпізнана, номер рядка, у якому знаходиться лексема та номер символа, з якого вона починається. Вся ця інформація дозволяє користувачу швидко знайти місце помилки, а також у цьому допомагає виділення у

текстовому редакторі необхідного місця кольором. Якщо ж тип лексеми розпізнаний то він додається у таблицю лексем, з вказанням поточного номеру лексеми та аналогічних до помилки параметрів. Формат таблиці використовується через те що це найпростіша структура яку можна швидко обробляти під час синтаксичного аналізу.

4.2.2 Синтаксичний аналіз

На наступному етапі, утворена після лексичного аналізу, таблиця лексем потрапляє до синтаксичного аналізатора, який проводить на ній синтаксичний аналіз. Метою синтаксичного аналізу є перевірка вірності порядку слідування лексем, на відміну від лексичного аналізатора, який перевіряв лише корректність кожної окремої лексеми. Для синтаксичного аналізу був обраний метод висхідного розбору, через простоту подальшого розширення грамматики та через її простоту, бо такі грамматики зазвичай не призводять до коллізій які не можуть бути вирішенні у рамках методу висхідного розбору.

Алгоритм висхідного розбору проходить у два етапи, спочатку, на основі граматики, формується матриця відношень лексем, яка дозволяє зрозуміти порядок слідування лексем та шляхи до переводу послідовності нетерміналів у термінали, у результати переходячи до корня граматики. Далі на вхід синтаксичному аналізатору по черзі передається кожен запис таблиці лексем і у разі виявлення повноцінної основи, разом з символами які були передані раніше, вона перетворюється у відповідний до граматики нетермінальний символ [6]. У разі закінчення таблиці лексем та неможливості перейти до корня граматики, відображається помилка з усією інформацією про останню лексему яку не вдалося перетворити у відповідний нетермінал. У синтаксичній помилці відображається та сама інформація що і у

лексичній, проте часто окрім останньої лексеми виділяється виділяється весь нетермінал, включаючи усі лексеми які в нього входили.

Під час синтаксичного аналізу, паралельно з перевіркою правильності порядку слідування лексем, формується так зване дерево розбору, яке і описує увесь шлях переведення лексем у кореневий нетермінал граматики. За допомогою дерева розбору і виконується етап генерації вихідного коду.

4.2.3 Генерація вихідного тексту

Останнім етапом трансляції і є генерації вихідного коду, у реалізованій системі у якості вихідного коду виступає опис структури експеримента за допомогою JSON. На вхід генератор отримує дерево синтаксичного розбору, яке крім вкладеності структури граматики мови експерименту ще й описує структуру вкладеності самих об’єктів експерименту, за допомогою чого, переходячи від кореня дерева далі, створюються відповідні об’єкти та вкладаються необхідним чином, згідно дерева розбору, фактично утворюючи один великий об’єкт класу эксперимента на використаній для реалізації системи мові програмування. Далі, за допомогою спеціальної бібліотеки Jackson, анотаціями якої було помічено всі об’єкти моделі експерименту, отриманий об’єкт переводиться у JSON формат та зберігається за вказаним у тексті опису експеримент шляхом.

4.2.4 Можливості розширення структури мови опису

Для додавання можливості легко розширювати граматику мови опису у структурі транслятора перед синтаксичним аналізом був доданий етап формування граматики з окремого файлу опису граматики, саме у цьому файлі і описана граматика мови, яка використовується синтаксичним аналізатором. Файл складається з двох частин: блоку опису граматики у формі близької до Бекуса-Наура і блоку опису типів за допомогою регулярних виразів.

Блок опису грамматики складається з терміналів, нетерміналів та типів, усі нетермінали вказуються у дужках, нетермінал та його опис розділені символом рівності з двома двокрапками, після цього знаходиться опис нетерміналу, який складається з усіх елементів граматики. Уся граматика розділена на окремі блоки, які описуються дуже схоже до натерміналів, проте мають ще й двокрапку біля дужок. Фрагмент граматики можна побачити на Рисунку 4.3.

```
<:сценарій:>
    <сценарій>::=<сценарій-термінал>\n<назва об’єкту>\n<початок руху>\n<напрям>\n<швидкість>\n<глибина>\n<шлях до моделі>
    <сценарій-термінал>::=сценарій
    <назва об’єкту>::=<назва об’єкту-термінал>TEXT
    <назва об’єкту-термінал>::=назва об’єкту
    <початок руху>::=<початок руху-термінал>POINT
    <початок руху-термінал>::=початок руху
    <напрям>::=<напрям-термінал>TEXT
    <напрям-термінал>::=напрям
    <швидкість>::=<швидкість-термінал>TEXT
    <швидкість-термінал>::=швидкість
    <шлях до моделі>::=<шлях до моделі-термінал>TEXT
    <шлях до моделі-термінал>::=шлях до моделі
<;сценарій;>
```

Рисунок 4.3 — Фрагмент блоку опису грамматики

Другий блок слугує для опису типів даних, які використовуються граматикою для опису своїх структур, на Рисунку 4.3 до типів відносяться ключові слова TEXT та POINT, які відображають будь який набір символів та точку. Блок опису типів даних зображений на Рисунку 4.4.

```
TEXT;;=^[a-zA-Z ’,.0-9A-Яа-яЁёЇїІіЄєҐґ':\\/]*$
POINT;;=^(\(-?[0-9]*.[0-9]*;-?[0-9]*.[0-9]*\) ?)+$
GROUP;;=^(((\[-?[0-9]*.[0-9]*\]|\[-?[0-9]*.[0-9]*(;-?[0-9]*.[0-9]*)+\]) ?)+)$
```

Рисунок 4.4 — Блок опису типів

На прикладі також видно тип даних який використовується для поля TTX у морського об’єкту. І представляє собою розширений його формат, як можна побачити, всі регулярні вирази починаються та закінчуються символами початку та кінця строки, це необхідно для того щоб коректно розбирати вхідний текст, так як перед виділенням типів він розбивається на окремі текстові лексеми і кожна лексема має повністю відповідати типу.

4.3 Висновки до розділу

У результаті виконаної роботи було програмно реалізовано транслятор та графічну частину програми.

Транслятор у свою чергу використовує метод висхідного розбору, так як він має легко розширюватись, це реалізовано за допомогою окремого файлу с граматикою, який використовується для того щоб визначати усі базові структурні елементи та скласти таблицю передування, за якою і проходить синтаксичний аналіз, без таблиці передування було б доволі важко зробити мову легкою для розширення, для розбиття вхідного тексту на лексеми використовувались регулярні вирази, які також є простими для зміни та розширення.

Створений графічний інтерфейс відповідає усім необхідностям, він є легким для сприйняття та має повну функціональність як для редагування тексту опису так і для його генерації.

5. РОБОТА КОРИСТУВАЧА З ПРОГРАМНОЮ СИСТЕМОЮ

У плані системи для користування розробленим програмним продуктом немає обмежень, за рахунок обраної мови. Фактично для використання додатку необхідно лише встановлення JRE, який можна скачати з офіційного сайту Oracle, восьмої або вищих версії та файли проекту. Пристрій, для якому виконання додатку, має мати розширення від 1024 до 768 та наявність вільної пам'яті від 200 Мб.

Графічний інтерфейс системи складається з двох окремих частин. Першою є генератор опису експерименту, а другий це редактор опису експерименту. Після завершення введення усі даних у генератор, та перевірки їх на коректність, відкривається вікно редактора тексту, після завершення роботи з ним відкривається вікно вибору файлу для збереження результуючого файлу, після чого виводиться або помилка з описом, або файл успішно зберігається та знову відкривається вікно для редагування опису експерименту.

5.1 Генератор опису експерименту

Генератор опису експерименту слугує для позбавлення користувача від необхідності власноруч вводити усі шаблонні слова, також представляє інтуїтивний та зручний інтерфейс для заповнення всіх необхідних полів структури. Генератор фактично повністю дублює сам опису експерименту, по своїй структурі.

При відкритті генератору користувач одразу потрапляє у форму для введення полів експерименту, а саме назву та опис (Рисунок 5.1).

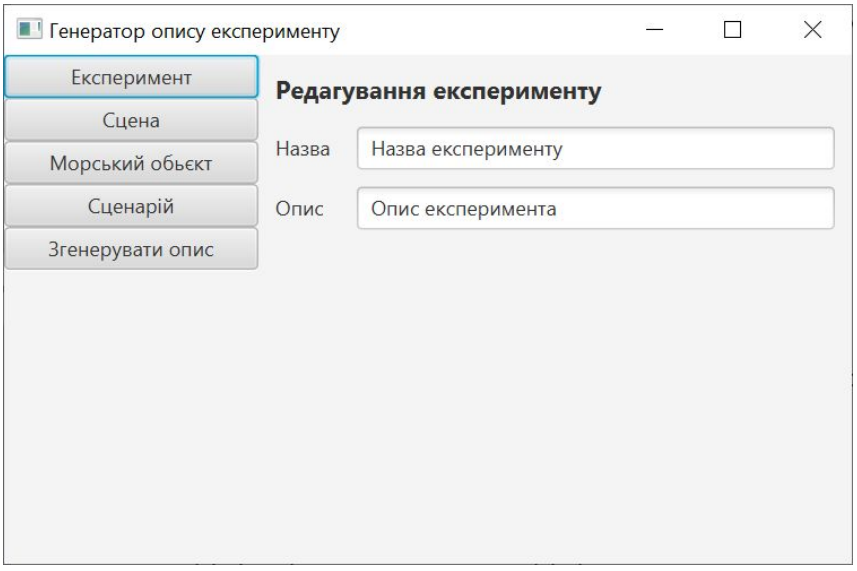

Рисунок 5.1 — Вікно форми для заповнення полів експерименту у генераторі опису

У цій формі крім самих полів для введення інформації є ще й окрема панель для переходу між формами різних об’єктів опису експерименту, впорядкованих аналогічно до мови опису. Після натискання на кнопку сцени, користувач потрапляє у вікно редагування сцени (Рисунок 5.2).

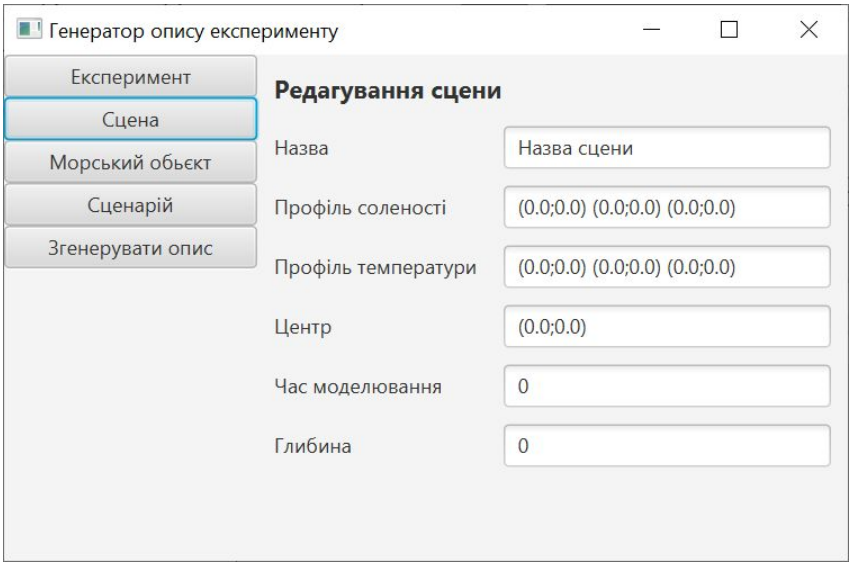

Рисунок 5.2 — Вікно форми для заповнення полів сцени у генераторі опису

У формі сцени видно ту саму панель що й у експерименту, проте зліва вже знаходяться поля експерименту, усі вони вводятся відповідно до формату мови опису, а профілі дозволяють задавати будь-яку кількість точок. Після нажаття на кнопку морського об'єкту відкривається його форма (Рисунок 5.3).

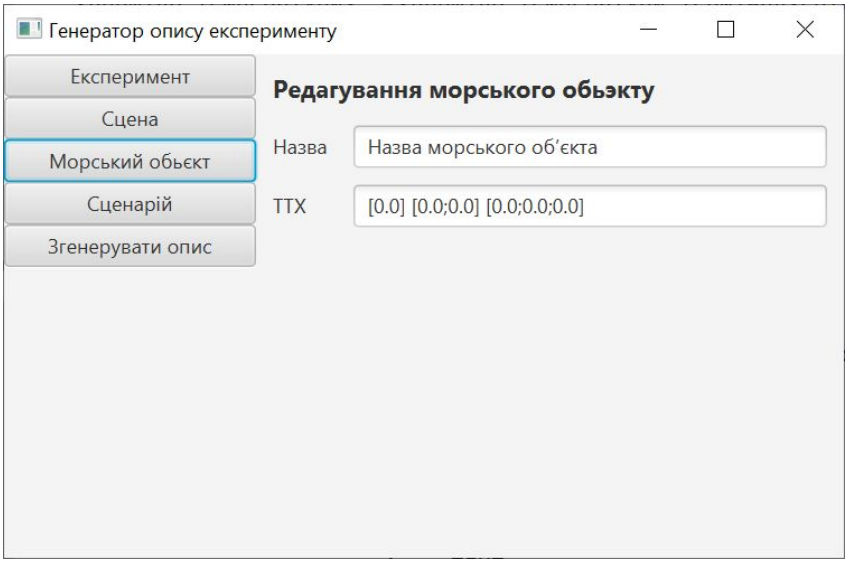

Рисунок 5.3 — Вікно форми для заповнення полів морського об’єкту у генераторі опису

Аналогічно до минулих прикладів у формі редагування морського об'єкту усі поля розміщені у тому ж порядку та мають той самий формат що і у мові опису. При нажатті на кнопку сценарію відкриється його форма (Рисунок 5.4)

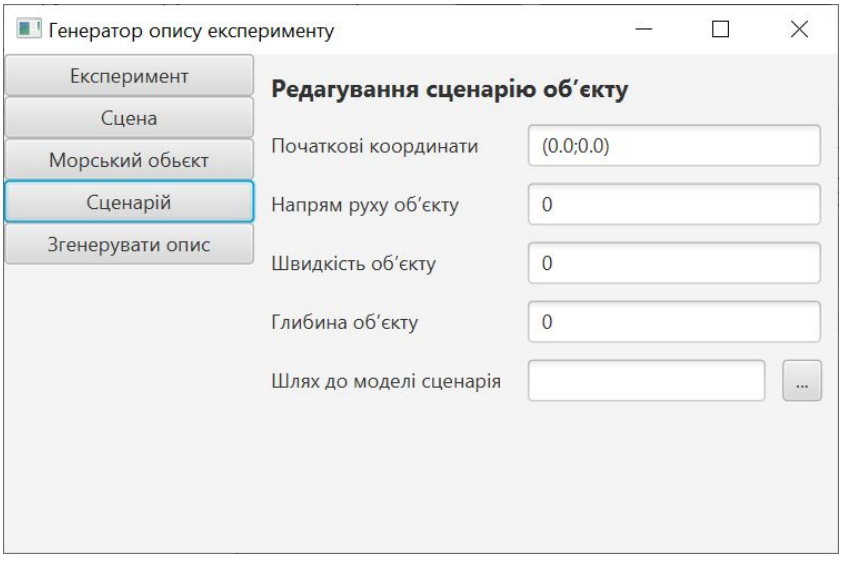

Рисунок 5.4 — Вікно форми для заповнення полів сценарія у генераторі опису

У формі сценарія також видно дефолтні значення у його полів, для всіх окрім шляху до моделі сценарія, це поле має бути завжди вказане, при нажатті на кнопку вибору файлу відкривається вікно вибору (Рисунок 5.5).

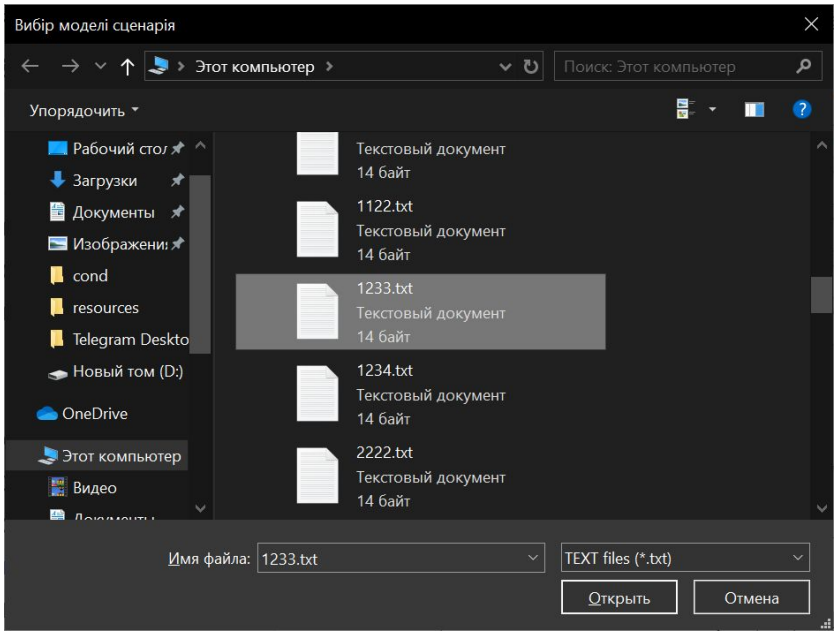

Рисунок 5.5 — Вікно для вибору файлу моделі для обробки сценарія

Після вибору файлу та нажаття на кнопку відкриття, шлях до цього файлу збережеться у поле шляху до моделі та знов відкриється форма для заповнення опису сценарію. (Рисунок 5.6)

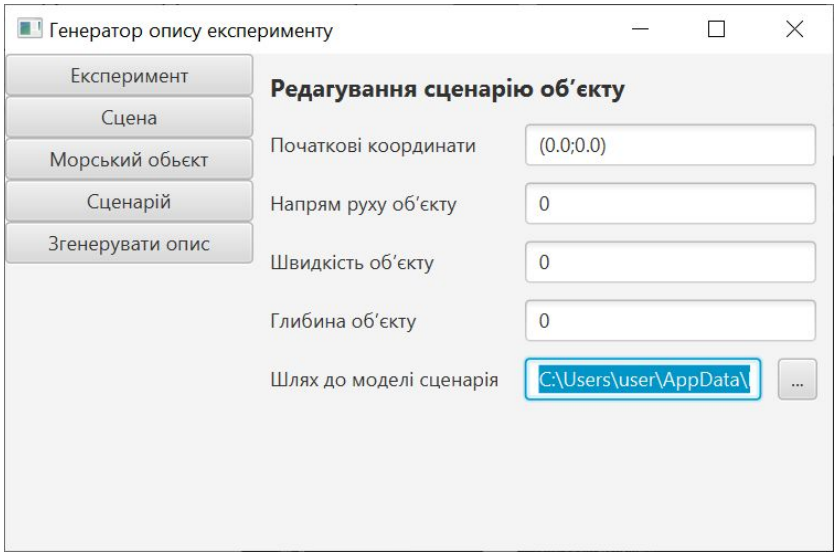

Рисунок 5.6 — Вікно форми для заповнення полів сценарія у генераторі опису з обраним файлом моделі

Далі у відкритому вікні після закінчення введення усіх полів усіх об'єктів можно нажати на кнопку генерації опису. У разі якщо шлях до моделі не був вказаний то відкриється вікно помилки (Рисунок 5.7)

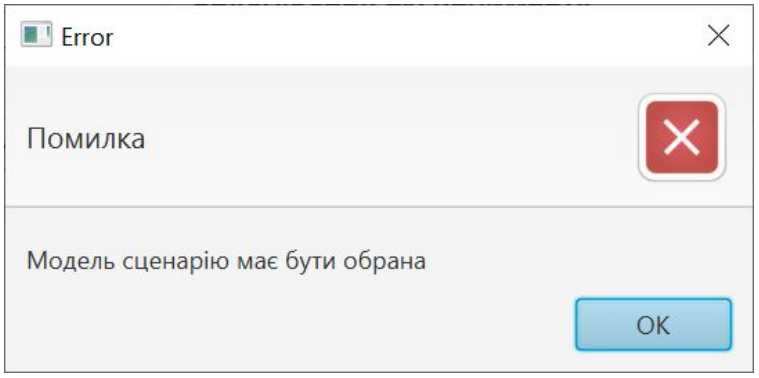

Рисунок 5.7 — Вікно помилки після спроби генерувати опис не вказуючи модель сценарія

Після виходу з вікна помилки необхідно лише обрати файл з моделлю та знов нажати на генерацію. Генератор опису надає зручний інтерфейс для генерації, згідно до задач.

5.2 Редактор опису експерименту

Після заповнення усіх полів форми, або навіть не заповнюючи, тоді у якості полів будуть використані дефолтні значення, можна нажати на кнопку введення та перейти у текстовий редактор. Текстовий редактор можна побачити на рисунку 5.8.

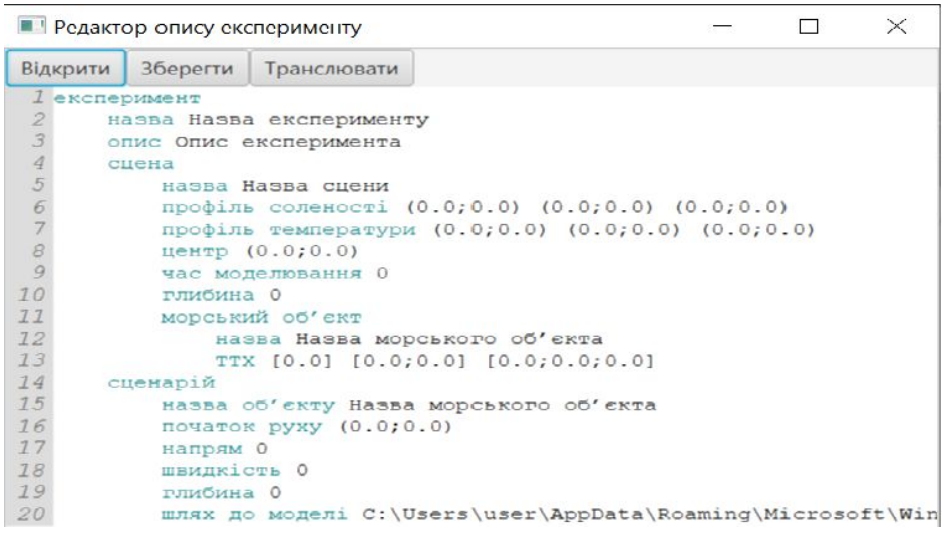

Рисунок 5.8 — Вікно текстового редактора

У текстовому редакторі можно обачити кнопки відкрити та зберегти, вони необхідні для відкриття вже написаних описів в редакторі або для збереження написаного у вигляді окремого файлу, також є кнопка для трансляції, яка дозволяє обрати файл у який буде збережено результуючий текст, після чого відбудется трансляція у нього та, якщо все введено верно, результат збережеться у вказаний файл. Основна частина редактора представляє собою поле для введення тексту опису, для спрощення сприйняття усі ключові слова, такі як назва об’єкту чи його поля, так як ці слова описують структуру то вони ніколи не мають змінюватися, проте у разі введення його власноруч та помилки у написанні кольорове виділення зніметься.

5.3 Висновки до розділу

У результаті виконання дипломної роботи був створений зручний та інтуїтивно зрозумілий графічний інтерфейс, який складається з генератора та редактору опису експерименту.

Генератор, у свою чергу, представляє собою вікно, яке розділено на окремі форми, для кожного з об’єктів, переключатися за якими можна завдяки панелі зліва від основної форми. Він також має влаштовану перевірку на наявність необхідних полів, через що неможливо згенерувати опис не вказавши шлях до моделі експерименту. А також, якщо користувачу не дуже важливі значення більшості полів вони мають своє значення за замовчуванням.

Створений редактор коду у свою чергу також надає зручний інтуїтивний інтерфейс для роботи з текстовим описом, а також дозволяє керувати вхідними та вихідними файлами, відкриваючи чи зберігаючи їх.

ВИСНОВКИ

У результаті виконання дипломної роботи було зроблено:

1. розроблена проста, легка для сприйняття та інтуїтивна по структурі мова опису гідроакустичних експериментів;
2. реалізовано транслятор для розробленої мови, який має функціонал для легко доповнення мови;
3. створена окрема середа для зручної роботи з описами експериментів та їх трансляцією;
4. автоматизовано друк шаблонного тексту за допомогою окремої форми для генерації опису сцени;

Розроблена окрема мова для опису гідроакустичних експериментів базується на практичності, простоті для розуміння та написання. Головною особливістю цієї мови є її зрозумілість для науковців, які навіть ніколи раніше не працювали з нею.

Мова опису експерименту включає в себе головні особливості предметно-орієнтованих мов програмування, що позбавляє її від недоліків мов загального призначення. Весь синтаксис повністю слугує опису елементам предметної області, не маючи чисто службових синтаксичних конструкції.

Разом з мовою був розроблений алгоритм для транслятора. Для реалізації легкості розширення мови для синтаксичного аналізу був обраний алгоритм висхідного розбору з додаванням файлу граматики, який можна легко доповнювати та розширювати, довго не розбираючись у структурі роботи системи.

Створений графічний інтерфейс виконує роль спрощення роботи з описами. Так форма для генерації опису успішно вирішує проблему написання великої кількості шаблонного коду, беручи цей обов’язок на себе. І навіть генеруючи дефолтні значення для усіх полів, тобто для створення найпростішого опису від користувача потрібно лише вибрати файл з моделлю сценарію та нажати на кнопку генерації.

Редактор опису, у свою чергу, спрощує візуальне сприйняття опису, тим самим дозволяє швидко орієнтуватися серед усіх об’єктів та знайти потрібне поле, яке потрібно замінити. Виділення кольором ключових слів також допоможе користувачу власноруч вводити опис структури.

Було отримано навички роботи з складними графічними інтерфейсами, зокрема створення форм вводу даних з можливістю будувати форми з великою кількістю рівнів вкладеності. Також був отриманий досвід з розробки текстових редакторів, розбиваючи текст на окремі частини та виділяючи їх особливим образом, паралельно з редагуванням тексту користувачем, тим самим отримавши навички роботи з багато проточними системами.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Лінгвістичне забезпечення сценаріїв і сцен моделюючих комплексів. Дипломна робота. Думчев

2. Java: эффективное программирование, 3-е издание. Джошуа Блох. 2018

3. Котлин в действии. Дмитрий Жемеров, С. Исакова. 2016

4. Ахо, А.В. Компиляторы: Принципы, технологии, инструментарий / А.В.Ахо, М.С. Лам, Р. Сети [и др.].– 2-е изд. : пер с анлг. – М.[и др.]: ИД Вильямс, 2010.– 1184 с.

5. Льюис, Ф. Теоретические основы проектирования компиляторов : пер. с англ. / Ф. Льюис, Д. Розенкранц, Р. Стирнз.– М. : Мир, 1979.– 656 с.

6. Транслятори: лексичний та синтаксичний аналізатори : навч. посіб. / В. М. Медведєва, В. А. Третяк; Нац. техн. ун-т України "Київ. політехн. ін-т". - К. : НТУУ "КПІ", 2012. - 148 с. - Бібліогр.: с. 146-148 - укр.

ДОДАТОК 1

Лінгвістичне забезпечення опису сцени проведення гідроакустичних експериментів

Специфікація

УКР.НТУУ«КПІ ім. Ігоря Сікорського»_ТЕФ_АПЕПС_ТІ6161_20Б

Аркушів 2

Київ – 2020

Позначення	Найменування	Примітки
Документація		
УКР.НТУУ «КПІ ім. Ігоря Сікорського»_ТЕФ_АПЕ ПС_ТІ6161_20Б	Записка Єдинак ТІ-61.docx	Пояснювальна записка
Компоненти		
УКР.НТУУ«КПІ ім. Ігоря Сікорського»_ТЕФ_АПЕ ПС_ТІ6161_20Б 12-1	View.kt	Модуль редактору коду
УКР.НТУУ«КПІ ім. Ігоря Сікорського»_ТЕФ_АПЕ ПС_ТІ6161_20Б 12-2	LexicalAnalyzer.kt	Модуль лексичного аналізатора
УКР.НТУУ«КПІ ім. Ігоря Сікорського»_ТЕФ_АПЕ ПС_ТІ6161_20Б 12-3	TerminalsTable.kt	Модуль таблиці терміналів
УКР.НТУУ«КПІ ім. Ігоря Сікорського»_ТЕФ_АПЕ ПС_ТІ6161_20Б 12-4	Divider.kt	Базовий модуль розділення вхідного тексту на лексеми

ДОДАТОК 2

Лінгвістичне забезпечення опису сцени проведення гідроакустичних експериментів

Текст програми

УКР.НТУУ«КПІ ім. Ігоря Сікорського»_ТЕФ_АПЕПС_ТІ6161_20Б 12-1

Аркушів 10

Київ – 2020

```
class MainView : View("Редактор опису експерименту") {
//змінна з текстом опису який редагується
  val text: String by param()
//класс для опису поля для роботи з кодом
  val codes = CodeArea(text).apply {
    minHeight = 400.0
    minWidth = 500.0
    paragraphGraphicFactory = LineNumberFactory.get(this)
// перехід по коллеціїї ключових слів та виділення їх у тексті, якщо вони є
    keywords.forEach { keyword ->

      val indexes = text.getStringsIndexes(keyword)
      indexes.forEach {
        setStyleClass(it.first, it.second, "command")
      }
    }
// при зміні тексту виконається код у фігурних дужках
    this.textProperty().onChange {
// всі старі стилі обнуляються
      setStyleClass(0, text.length, "defaul")
// ключові слова знов виділяються
      keywords.forEach { keyword ->

        val indexes = text.getStringsIndexes(keyword)
        indexes.forEach {
          setStyleClass(it.first, it.second, "command")
        }
      }
    }
  }
// шлях до обраного файлу моделі сценрія
  var scenarioPath = ""
```

```
  override val root = scrollpane {
    borderpane {
      top = hbox {
        button("Відкрити") {
          setOnAction {
            try {
// відкриття файлу з екстом
              val fileChooser = chooseFile("dsfdsf", arrayOf(FileChooser.ExtensionFilter("TEXT
files (*.txt)", "*.txt")))
              codes.replaceText(fileChooser.first().readText())
            } catch (ex: Exception) {
              println(ex.message)
            }
          }
        }
        button("Зберегти") {
          setOnAction {
            try {
//зберігання опису у файл з текстом
              val fileChooser = chooseFile("dsfdsf", arrayOf(FileChooser.ExtensionFilter("TEXT
files (*.txt)", "*.txt")), mode = FileChooserMode.Save)
              fileChooser.first().writeText(codes.text)
            } catch (ex: Exception) {
              println(ex.message)
            }
          }
        }
        button("Транслювати") {
          setOnAction {
// зберігання результату трансляції у файл
            val fileChooser = chooseFile("dsfdsf", arrayOf(FileChooser.ExtensionFilter("TEXT files
(*.txt)", "*.txt")), mode = FileChooserMode.Save)
            File(DEFAULT_CODE_PATH).writeText(codes.text)
```

```
                val model = Model()
                val controller = Controller(model)
                val result = try {
//виклик транслятора
                    controller.run()
                } catch (ex: Exception) {
                    alert(Alert.AlertType.ERROR, "Помилка трансляції", ex.message)
                    null
                }
                if (result != null) {
                    var final = ""
//переведення результату трансляції у необхідний формат
                    val res = jacksonObjectMapper().readValue(result, MutableMap::class.java) as
MutableMap<String, MutableMap<String, Any>>
                    final = jacksonObjectMapper().writeValueAsString(res)
                    fileChooser.first().writeText(final)
                } else
                    fileChooser.first().writeText("Помилка")
            }
        }
    }
    center = codes
  }
}
// утилітний метод для отрминня індексів ключових слів
  fun String.getStringsIndexes(substring: String): MutableList<Pair<Int, Int>> {
    val result = mutableListOf<Pair<Int, Int>>()

    var lastFindIndex = 0
    while (true) {
      val current = "(^|\\n)\\s*$substring".toRegex().find(this, lastFindIndex)?.range
      if (current == null) return result
      lastFindIndex = current.endInclusive + 1
```

```
            result.add(current.start to current.endInclusive + 1)
        }
    }
}
// класс для стилів ключових слів та звичайного тексту
class Keywords : Stylesheet() {
    companion object {
        val command by cssclass()
        val defaul by cssclass()
    }

    init {
        command {
            fill = Color.TEAL
        }
        defaul {
            fill = Color.BLACK
        }
    }
}
```



```
public class LexicalAnalyzer {
    CodeText code;
    LexemeTypes lexemeTypes;
    TablesManager tables;
    SpecialLexemeTypes specTypes;
    public LexicalAnalyzer() throws Exception{
        code = new CodeText(TextReader.code().get());
        lexemeTypes = new LexemeTypes(TextReader.grammar().get());
        specTypes = new SpecialLexemeTypes(TextReader.regexGrammar().get());
        tables = new TablesManager(code);
    }
    public void analyse(){
```

```
// перебів усіх вхідних лексем та їх зберігання
      while(code.hasNext()){
         code.goToNextLexeme();
         saveLexeme();
      }
   }
//метод для зберігання лексем
   private void saveLexeme(){
      boolean hasLexType = lexemeTypes.hasType(code.getCurLexeme());
      if(hasLexType){
         tables.addCurLexeme();
         return;
      }
      String specType = specTypes.getType(code.getCurLexeme());
      boolean isSpecType = specType!=null;
      if(isSpecType){
         tables.addCurLexeme(specType);
         return;
      }
      if(code.getCurLexeme().equals("")||code.getCurLexeme().equals(" ")) return;
      throw new NotLexemeException(code.getCurLexeme(),code.getLineNum(),code.getLexemeNum());
   }
   public TablesManager getTables(){
      return tables;
   }
}
```



```
public class TerminalsTable {
   private List<Terminal> terminalsTable;
// класс для керування повним набором таблиць, які отримуються  в результаті лексичного аналізу
   private TablesManager tables;

   public TerminalsTable(TablesManager tables) {
```

```
        this.tables = tables;
        terminalsTable = new ArrayList<>();
        while (tables.hasNext()) {
            tables.goNext();
            terminalsTable.add(new Terminal(tables.get().lexeme()));
        }
        tables.goTo(-1);
        for (int i = 0; i < terminalsTable.size() - 1; i++) {
            if (terminalsTable.get(i).getName().equals("{") && terminalsTable.get(i +
1).getName().equals("\\n")) {
                terminalsTable.set(i, new Terminal("{\\n"));
                terminalsTable.remove(i + 1);
            }
        }
    }

    public Terminal peek() {
        if (isEmpty())
            return new Terminal();
        return terminalsTable.get(0);
    }

    public Terminal pop() {
        if (isEmpty())
            return new Terminal();
        tables.goNext();
        return terminalsTable.remove(0);
    }

    public String getSpecTypeName() {
        if (tables.getIndex() > -1)
            if (tables.get().getSpecType() != null) {
                return tables.get().getSpecType();
```

```
        }
        return null;
    }

    public String getSpecTypeValue() {
        if (tables.getIndex() > -1)
            if (tables.get().getSpecType() != null) {
                return tables.get().name();
            }
        return null;
    }

    public boolean isEmpty() {
        return terminalsTable.isEmpty();
    }

    @Override
    public String toString() {
        return "TerminalsTable{" +
                "terminalsTable=" + terminalsTable +
                '}';
    }
}
```



```
public abstract class Divider {
    String selector;
    String lineEnd;
    String splitRegex;
    String[] separators = new String[]{};
    String[][] regexSeparators;
    String[][] multiRegexSeparators = new String[][]{};
    String startLine;
    String stopLine;
```

```
    boolean isStarted = false;

    private static List<List<String>> splitText;
    private static String curLine;
// метод ддя розділення вхідного тексту на лексеми
    public List<List<String>> splitText(final List<String> text) {
        splitText = new ArrayList<>();
        for (String line : text) {
            if(!line.equals(EMPTY)) {
                if (line.equals(stopLine))
                    return splitText;
                if (isStarted) {
                    curLine = line;
                    addEnd();
                    selectSimpleSeparators();
                    selectRegexSeparators();
                    selectMultiRegexSeparators();
                    saveLine();
                }
                if (line.equals(startLine))
                    isStarted = true;
            }
        }
        return splitText;
    }
// додання роздільників для розбиття вхідного тексту на лексеми
    private void selectSimpleSeparators() {
        for (String separator : separators) {
            curLine = curLine.replaceAll(separator, selector + separator + selector);
        }
    }

    private void selectRegexSeparators() {
```

```
        for (String[] regexSeparator : regexSeparators) {
            curLine = curLine.replaceAll(regexSeparator[0], regexSeparator[1]);
        }
    }
// метод для виділення лексем за допомогою регулярних виразів
    private void selectMultiRegexSeparators(){
        for(String[] multiRegexSeparator : multiRegexSeparators){
            Pattern pattern = Pattern.compile(multiRegexSeparator[0]);
            Matcher matcher = pattern.matcher(curLine);
            while(matcher.find()){
                curLine = curLine.replaceAll(multiRegexSeparator[0], multiRegexSeparator[1]);
                matcher = pattern.matcher(curLine);
            }
        }
    }

    private void addEnd() {
        curLine = curLine + lineEnd;
    }

    private void saveLine() {
        splitText.add(Arrays.asList(removeEmpty(curLine.split(splitRegex))));
    }
}
```

ДОДАТОК 3

Лінгвістичне забезпечення опису сцени проведення гідроакустичних експериментів

Опис програмного модулю

УКР.НТУУ«КПІ ім. Ігоря Сікорського»_ТЕФ_АПЕПС_ТІ6161_20Б 13-1

Аркушів 8

Київ – 2020

АНОТАЦІЯ

Представлена програмна система створена для надання можливості лінгвістичного опису гідроакустичних експериментів, а також для спрощення роботи з розробленим описом.

Програмна система представляє собою транслятор для переведення опису експерименту у формат JSON, а також інтерфейс для редагування та генерації описів. Ця система розроблена за допомогою мови загального призначення Kotlin, основана на JVM.

Рекомендовано запускати її на платформі операційної системи Windows, проте цілком можливий запуск і на будь-яких інших.

Перевагою розробленої системи є її зручність у використанні та гнучкість. У будь який момент розробник доволі легко та швидко може розширити систему, змінюючи лише файл з граматикою мови.

ЗМІСТ

ЗАГАЛЬНІ ВІДОМОСТІ

Аналогічно назві дипломної роботи, розроблена система має назву “Система лінгвістичного забезпечення опису сцени проведення гідроакустичних експериментів”.

Система представляє собою додаток, який можливо скомпілювати та запустити під будь-яку систему, використовуючи JRE. Проте при розробці система в основному орієнтована на Windows.

Представлена система була реалізована на мові Kotlin, та використовує для своєї роботи бібліотеку TornadoFx.

ФУНКЦІОНАЛЬНЕ ПРИЗНАЧЕННЯ

Призначенням створеної програми є трансляція файлу з вхідною мовою опису експерименту у вихідний файл формату JSON, що не має ніяких обмежень. А також система має надавати зручний інтерфейс для збереження, редагування та додавання файлів з текстами граматики, надавати зручний і зрозумілий інтерфейс для забезпечення максимальної інтуїтивності сприйняття користувачем опису експерименту. Програма повинна також автоматизувати введення шаблонного тексту опису користувачем за допомогою окремої форми, яка повністю дублює структуру мови, а також надає додаткову валідацію введених даних.

Для трансляції був обраний алгоритм висхідного розбору для вирішення проблеми легкості розширення граматики мови. Для лексичного аналізу використовувався метод регулярних виразів.

У результаті роботи програма відправляє системі моделювання гідроакустичних процесів всі необхідні дані для проведення експерименту у форматі Json.

ОПИС ЛОГІЧНОЇ СТРУКТУРИ

Створена програма структурно складається з декількох незалежних частин і перехід між ними завжди відбувається за одним і тим сценарієм.

Спочатку відкривається вікно генерацію опису експерименту, у яку користувач вводить усі необхідні значення полів та обирає файл з моделлю для обробки сценарія, після чого переходить у вікно редагування згенерованого опису експерименту. У разі якщо модуль сценарію не була обрана то виникне вікно з помилкою, користувач не зможе продовжити робоу не обравни модель сценарію.

Далі відкривається редактор опису експерименту. У ньому користувач може відкрити інший сценарій, якщо згенерований його не влаштовує, або редагувати вже згенерований. У полі для редагування виділені усі ключові слова для спрощення сприйняття користувачем інформації та структури експерименту, усі ключові слова завжди знаходяться на початку кожного рядка. Після закінчення редагування користувач також може зберегти введений опис експерименту у файл. Коли користувач закінчив ввід усіх необхідних даних він може виконати трансляцію введеного тексту у вихідний Json та обрати місце де він буде збережений.

Трансляція відбувається у три етапи. Спочатку вхідний текст розбивається на лексеми під час лексичного аналізу. Це відбувається за допомогою спеціально введених регулярних виразів та ключових слів. Далі до синтаксичного аналізатора потрапляють таблиці лексем, впорядковані так само як у вхідному файлі. Синтаксичний аналіз відбувається за допомогою алгоритму висхідного розбору, який на основі введеної граматики будує дерево. На останньому етапі дерево транслюється у вихідний текст.

Після закінчення роботи вихідний текст потрапляє у систему проведення експерименту та слугає початковими значеннями для нього.

ВИКОРИСТАНІ ТЕХНІЧНІ ЗАСОБИ

Для реалізації програми була використана мова програмування Kotlin та середа розробки Intellij IDEA, розроблена компанією JetBrains. Для створення графічного інтерфейсу було використано TornadoFx.

Для запуску програми користувачу необхідно використовувати JRE 8 або вищі версії.

ВХІДНІ ТА ВИХІДНІ ДАНІ

Вхідні дані:

1. назва та опис експерименту
2. параметри сцени
3. параметри морського об’єкту
4. параметри сценарію та модель для його обробки

Вихідні дані:

1. Json з описом гідроакустичного експерименту